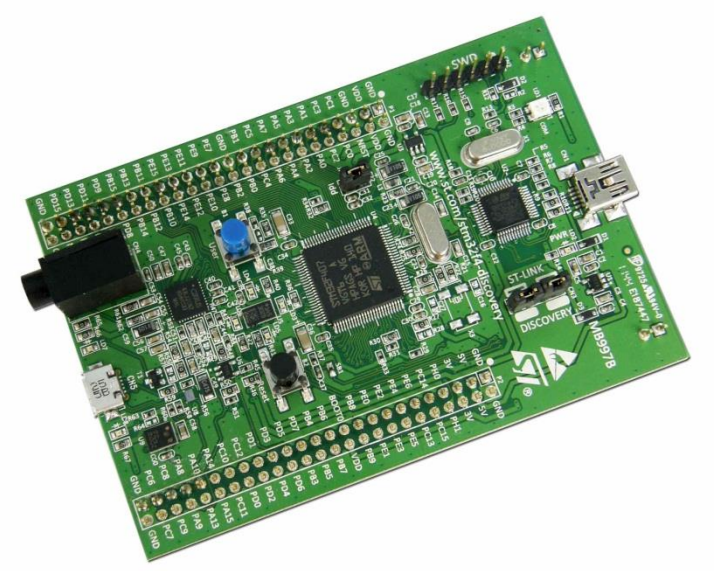
Politechnika Poznańska, Instytut Automatyki i Inżynierii Informatycznej

Podstawy technik mikroprocesorowych

Ćwiczenia laboratoryjne ADC/DAC



Marek Kraft, Michał Fularz

2013-11-20

1. Wstęp teoretyczny.

Na zajęcia obowiązuje znajomość materiału zaprezentowanego na wykładzie.
Przetworniki cyfrowo-analogowe (DAC) i analogowo-cyfrowe (ADC) są pośredniczącymi blokami funkcjonalnymi mikrokontrolera, stosowanymi do konwersji pomiędzy sygnałami analogowymi (charakteryzującymi zjawiska występujące w przyrodzie i otrzymywanych na drodze pomiarowej) a układami cyfrowymi (np. mikrokontrolery, komputery itp.) wykorzystującymi w swoim działaniu dyskretne wartości sygnałów.

2. Przebieg ćwiczenia.

a. Tworzenie projektu.

Utworzyć nowy projekt (lub skorzystać z programu z poprzednich zajęć). W zakładce ***Repositories*** dodać następujące moduły:

- ADC – funkcje obsługi przetwornika analogowo-cyfrowego (dodaje pliki stm32f4xx_adc.h oraz stm32f4xx_adc.c),
- DAC – funkcje obsługi przetwornika cyfrowo-analogowego (dodaje pliki stm32f4xx_dac.h oraz stm32f4xx_dac.c).

W oknie edytora kodu otworzyć załączone pliki (aby je podświetlić w oknie ***Components*** wybrać odpowiedni moduł (ADC lub DAC)). Zwrócić uwagę na:

- Exported types (stm32f4xx_adc.h – ADC_InitTypeDef, ADC_CommonInitTypeDef, stm32f4xx_dac.h – DAC_InitTypeDef),
- Exported constants,
- Exported functions (stm32f4xx_adc.h – ADC_Init, ADC_CommonInit, ADC_RegularChannelConfig, ADC_Cmd, ADC_SoftwareStartConv, ADC_GetFlagStatus, ADC_GetConversionValue; stm32f4xx_dac.h – DAC_Init, DAC_SetChannel**X**Data, DAC_Cmd).

W plikach stm32f4xx_adc.c oraz stm32f4xx_dac.c znajduje się krótki opis funkcjonalności danego modułu (na początku pliku, w komentarzu), a także definicje funkcji wraz z ich krótki opisem (indeksowanym przez IDE i wyświetlanym po umieszczeniu kursora na funkcji i naciśnięciu klawisza F2).

Do pliku z funkcją **main** programu dodać odpowiednie biblioteki (#include ”stm32f4xx_adc.h” oraz #include ”stm32f4xx_dac.h”).

b. Konfiguracja przetwornika ADC.

Przed przystąpieniem do konfiguracji przetwornika ADC, konieczne jest skonfigurowanie i włączenie generatora sygnału zegarowego, a następnie doprowadzenia sygnału zegarowego do układów peryferyjnych, które mają być wykorzystane:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);     // zegar
dla portu GPIO z którego wykorzystany zostanie pin jako wejście ADC
(PA1)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);       // zegar
dla modułu ADC1
```

Domyślnie, po włączeniu zasilania, do bloków funkcjonalnych nie jest doprowadzony sygnał zegarowy, co umożliwia obniżenie poboru prądu.

Konfiguracji samych portów dokonuje się wykorzystując funkcję GPIO_Init. Funkcja ta jako parametry przyjmuje nazwę portu, który ma zostać zainicjalizowany oraz strukturę typu GPIO_InitTypeDef. Poszczególne ustawienia portu wprowadza się inicjalizując pola struktury.

```
GPIO_InitTypeDef GPIO_InitStructure;

//inicjalizacja wejścia ADC
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Dla przypomnienia:

- Pole GPIO_Pin zawiera informację o tym, które wyprowadzenia portu maja zostać zainicjalizowane.
- Pole GPIO_Mode zawiera informację o trybie działania wyprowadzenia
  - GPIO_Mode_IN - wejście binarne
  - GPIO_Mode_OUT - wyjście binarne
  - GPIO_Mode_AF - funkcja alternatywna
  - GPIO_Mode_AN - wejście/wyjście analogowe
- Pole GPIO_OType zawiera informację o typie wyjścia
  - GPIO_OType_PP - wyjście komplementarne
  - GPIO_OType_OD - wyjście z otwartym drenem
- Pole GPIO_Speed informuje o maksymalnej dozwolonej prędkości przełączania wyprowadzeń
  - GPIO_Speed_2MHz
  - GPIO_Speed_25MHz

- GPIO_Speed_50MHz
- GPIO_Speed_100MHz

- Pole GPIO_PuPd zawiera konfigurację rodzaju podciągania wyprowadzenia
  - GPIO_PuPd_NOPULL - brak podciągania
  - GPIO_PuPd_UP - podciąganie do napięcia zasilania
  - GPIO_PuPd_DOWN - ściąganie do masy

Proszę zwrócić uwagę na linijkę:

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
```

Konfiguruje ona wybrane wyprowadzenia jako wejścia lub wyjścia współpracujące z sygnałami analogowymi.

Aby skonfigurować sam przetwornik ADC należy, podobnie jak w przypadku portów wejścia/wyjścia, zainicjalizować pola dedykowanej dla tego układu peryferyjnego struktury.
Wspólna konfiguracja dla wszystkich układów ADC:

```
ADC_CommonInitTypeDef ADC_CommonInitStructure;
// niezależny tryb pracy przetworników
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
// zegar główny podzielony przez 2
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
// opcja istotna tylko dla tryby multi ADC
ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
// czas przerwy pomiędzy kolejnymi konwersjami
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles;
ADC_CommonInit(&ADC_CommonInitStructure);
```

Konfiguracja danego przetwornika ADC:

```
ADC_InitTypeDef ADC_InitStructure;
//ustawienie rozdzielczości przetwornika na maksymalną (12 bitów)
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
//wyłączenie trybu skanowania (odczytywać będziemy jedno wejście ADC
//w trybie skanowania automatycznie wykonywana jest konwersja na wielu
//wejściach/kanałach)
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
//włączenie ciągłego trybu pracy
ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
//wyłączenie zewnętrznego wyzwalania
//konwersja może być wyzwalana timerem, stanem wejścia itd. (szczegóły w
//dokumentacji)
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;
//wartość binarna wyniku będzie podawana z wyrównaniem do prawej
//funkcja do odczytu stanu przetwornika ADC zwraca wartość 16-bitową
//dla przykładu, wartość 0xFF wyrównana w prawo to 0x00FF, w lewo 0x0FF0
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
//liczba konwersji równa 1, bo 1 kanał
```

```
ADC_InitStructure.ADC_NbrOfConversion = 1;
// zapisz wypełnioną strukturę do rejestrów przetwornika numer 1
ADC_Init(ADC1, &ADC_InitStructure);
```

W następnej kolejności należy skonfigurować wybrany kanał ADC:

```
ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_84Cycles);
```

W przykładzie powyżej konfigurujemy pierwszy kanał przetwornika ADC1 do samodzielnej pracy, ustawiając czas próbkowania na 3 cykle sygnału zegarowego. Można teraz uruchomić przetwornik ADC:

```
ADC_Cmd(ADC1, ENABLE);
```

Odczyt wartości poprzez odpytywanie flagi zakończenia konwersji wykonujemy następująco:

```
ADC_SoftwareStartConv(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
ADC_Result = ADC_GetConversionValue(ADC1);
```

Zadanie:

Korzystając z dodatkowej płytki połączyć, za pomocą przewodu, sygnał z potencjometru z wejściem skonfigurowanego przetwornika ADC. Stworzyć zmienną globalną, do której przypisany będzie rezultat konwersji analogowo-cyfrowej. Bazując na opisie w punkcie dotyczącym programu **STM Studio** zaimportować stworzoną zmienną i wyświetlić ja na wykresie typu Bar Graph. Zmieniając położenie drążka potencjometru obserwować zmiany wartości na wykresie. Jaka jest maksymalna możliwa do uzyskania wartość? Czy odpowiada to ustawionej rozdzielczości przetwornika?

c. Konfiguracja przetwornika DAC.

Przed przystąpieniem do konfiguracji przetwornika DAC, konieczne jest skonfigurowanie i włączenie generatora sygnału zegarowego, a następnie doprowadzenia sygnału zegarowego do układów peryferyjnych, które mają być wykorzystane:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA , ENABLE);    // zegar
dla portu GPIO z którego wykorzystany zostanie pin jako wyjście DAC
(PA4)
RCC_APB2PeriphClockCmd(RCC_APB2Periph_DAC, ENABLE); // zegar dla
modułu DAC
```

Domyślnie, po włączeniu zasilania, do bloków funkcjonalnych nie jest doprowadzony sygnał zegarowy, co umożliwia obniżenie poboru prądu.

Konfiguracji samych portów dokonuje się wykorzystując funkcję GPIO_Init. Funkcja ta jako parametry przyjmuje nazwę portu, który ma zostać zainicjalizowany oraz

strukturę typu GPIO_InitTypeDef. Poszczególne ustawienia portu wprowadza się inicjalizując pola struktury.

```
GPIO_InitTypeDef GPIO_InitStructure;

//inicjalizacja wyjścia DAC
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOA, &GPIO_InitStructure);
```

Aby skonfigurować sam przetwornik DAC należy, podobnie jak w przypadku portów wejścia/wyjścia, zainicjalizować pola dedykowanej dla tego układu struktury.

```
DAC_InitTypeDef DAC_InitStructure;
//wyłączenie zewnętrznego wyzwalania
//konwersja może być wyzwalana timerem, stanem wejścia itd.
(szczegóły w dokumentacji)
DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
//nast. 2 linie - wyłączamy generator predefiniowanych przebiegów
//wyjściowych (wartości zadajemy sami, za pomocą odpowiedniej
funkcji)
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude =
DAC_LFSRUnmask_Bit0;
//włączamy buforowanie sygnału wyjściowego
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
```

Można teraz uruchomić przetwornik DAC:

```
DAC_Cmd(DAC_Channel_1, ENABLE);
```

Przetwornik ma rozdzielczość 12 bitów, więc ustawienie maksymalnej wartości odbywa się następująco (jako wartości podajemy liczbę 16-bitową wyrównaną do prawej):

```
DAC_SetChannel1Data(DAC_Align_12b_R, 0xFFF);
```

Korzystając z powyższej funkcji możliwe jest zmienianie wartości generowanej przez moduł DAC.

Zadanie:

Napisać program, który co określny czas (np. 0,5 sekundy) zmienia wartość generowaną prze przetwornik DAC. Podłączyć płytkę do oscyloskopu (pod nadzorem prowadzącego) i zweryfikować poprawność działania.

d. Obsługa programu STM Studio.

Ze strony:

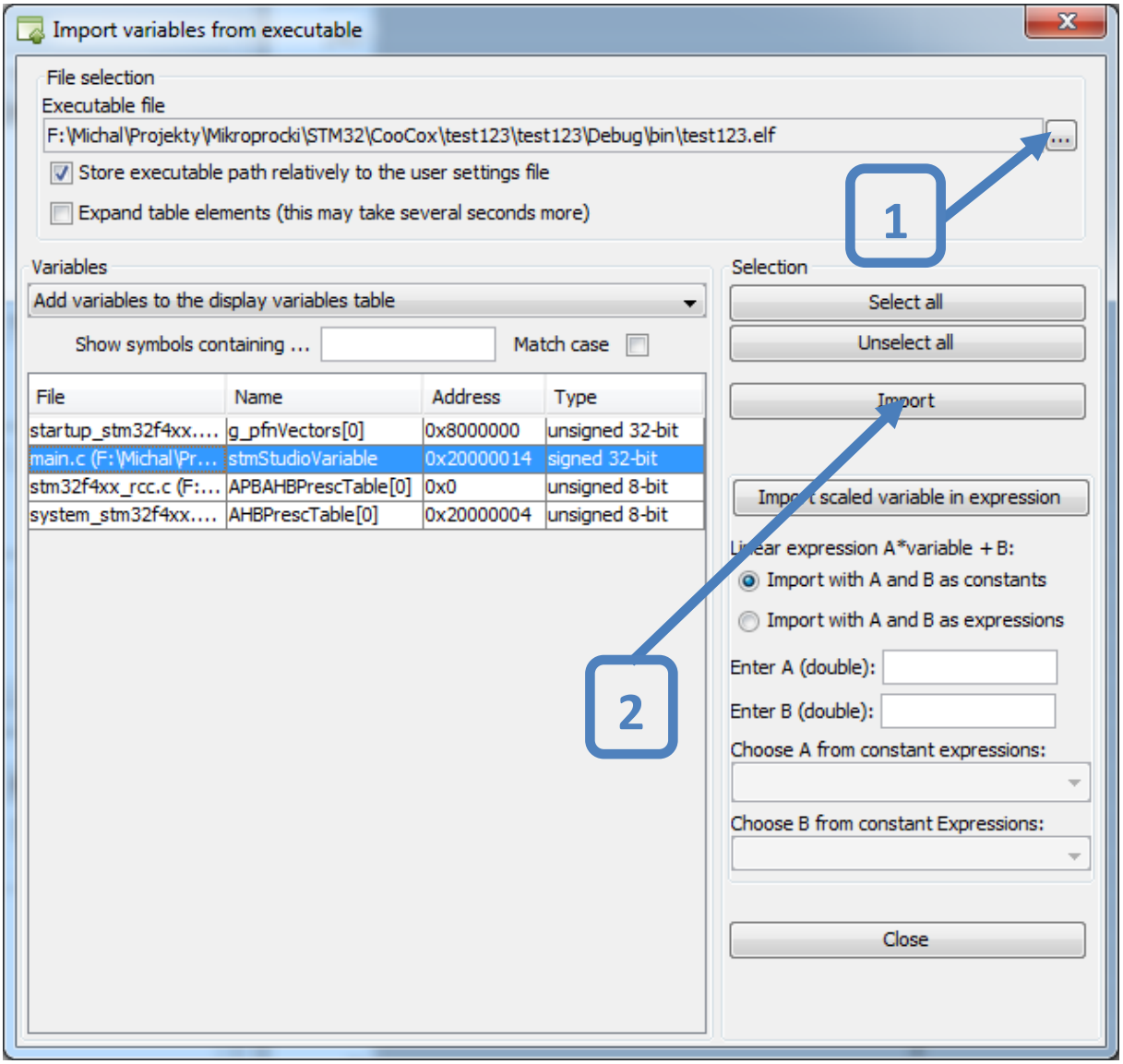
http://www.st.com/web/en/catalog/tools/PF251373

pobrać oprogramowanie STM Studio (aktualna wersja: 3.3) i zainstalować na komputerze. W przypadku braku wirtualnej maszyny Java pobrać zgodnie z sugestiami instalatora programu STM Studio (zainstalować wersję dla systemu 32 bitowego).

Po uruchomieniu programu STM Studio z menu ***File*** wybrać ***Import variables***. W oknie przedstawionym poniżej wybrać ścieżkę dostępu do pliku wgrywanego do procesora (w folderze ***Debug*** w lokalizacji projektu) [1]. Następnie wybrać zmienne do podglądu (w przykładowym oknie ***stmStudioVariable***) i nacisnąć przycisk ***Import*** [2].

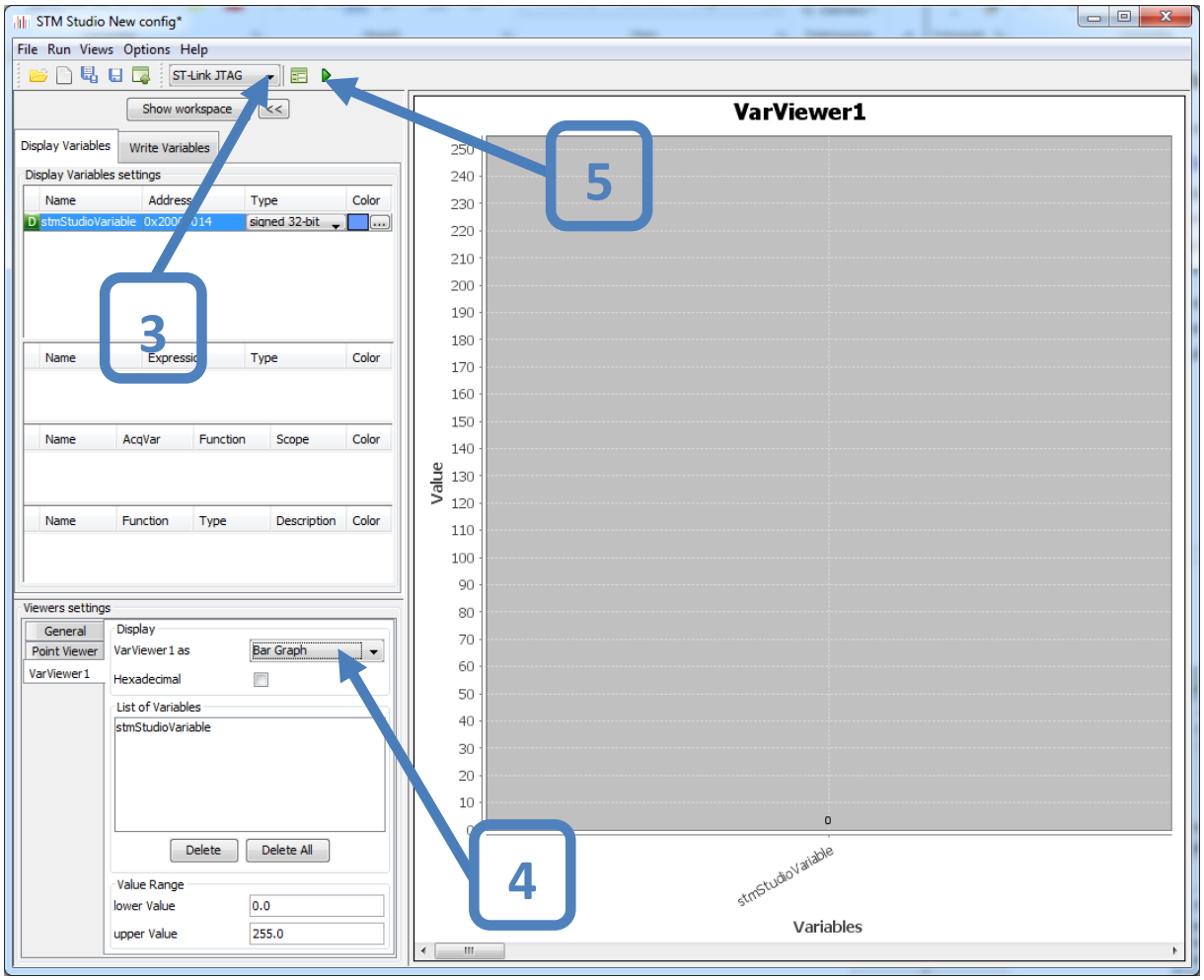
UWAGA!!!

Zmienne wybierane do podglądu muszą być globalne.



Po załadowaniu zmiennych do podglądu z menu ***Options*** wybrać ***Acquisition Settings*** i odznaczyć opcję ***Log to file***. Z rozwijanego menu wybrać ***ST-link SWD*** [3]. Dodać stworzoną zmienną do wyświetlania (prawym klawiszem myszy na dodaną zmienną i

z pojawiającego się menu wybrać ***Send to->VarViewer1***. Zmienić typ wykresu na ***Bar Graph*** [4]. Po wgraniu programu przez CooCox nacisnąć przycisk ***Play*** [5].



e. Zadania do samodzielnej realizacji.

- Korzystając z przetwornika ADC powiązać sposób wyświetlania diod z wartością mierzonego napięcia (np. wraz ze wzrostem napięcia zapalają się dodatkowe diody).
- Przeskalować surową wartość pozyskiwana z przetwornika przyjmując, że mierzona jest wartość temperatury w zakresie -30 do 50 stopni.
- Połączyć wyjście przetwornika DAC z ADC. Napisać program, który w regularnych odstępach czasu (np. co 200ms) zwiększa wartość generowanego napięcia (od 0 do maksymalnej wartości, w np. 20 krokach). Uruchomić program STM Studio i zweryfikować poprawność działania programu.
- Zmodyfikować program, aby korzystał, z dwóch kanałów ADC (do jednego podłączyć sygnał DAC, do drugiego sygnał z potencjometru). W zależności od położenia pokrętła potencjometru zmieniać prędkość modyfikacji sygnału DAC.